Acceleration of Bulk Memory Operations in a Heterogeneous Multicore Architecture

JongHyuk Lee
University of Houston
Houston, TX 77004, USA
jonghyuk.lee@daum.net

Ziyi Liu
University of Houston
Houston, TX 77004, USA
brian.zy.liu@gmail.com

Xiaonan Tian
University of Houston
Houston, TX 77004, USA
daniel.xntian@gmail.com

Dong Hyuk Woo
Intel Labs
Santa Clara, CA 95054, USA
dong.hyuk.woo@intel.com

Weidong Shi
University of Houston
Houston, TX 77004, USA
larryshi@ymail.com

Dainis Boumber
University of Houston
Houston, TX 77004, USA
dainis.boumber@gmail.com

ABSTRACT

In this paper, we present a novel approach of using the integrated GPU to accelerate conventional operations that are normally performed by the CPUs, the bulk memory operations, such as memcpy or memset. Offloading the bulk memory operations to the GPU has many advantages, i) the throughput driven GPU outperforms the CPU on the bulk memory operations; ii) for on-die GPU with unified cache between the GPU and the CPU, the GPU private caches can be leveraged by the CPU for storing moved data and reducing the CPU cache bottleneck; iii) with additional lightweight hardware, asynchronous offload can be supported as well; and iv) different from the prior arts using dedicated hardware copy engines (e.g., DMA), our approach leverages the exiting GPU hardware resources as much as possible. The performance results based on our solution showed that offloaded bulk memory operations outperform CPU up to 4.3 times in micro benchmarks while still using less resources. Using eight real world applications and a cycle based full system simulation environment, the results showed 30% speedup for five, more than 20% speedup for two of the eight applications.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General—System architectures; C.1.3 [**Processor Architectures**]: Other Architecture Styles—Heterogeneous (hybrid) systems

General Terms

Design, Performance

Keywords

Bulk memory operation, GPU, Heterogeneous multicore architecture, SIMD

1. INTRODUCTION

General purpose computing on GPUs (GPGPUs) has become a widely adopted architectural solution to build high performance computing systems, e.g. three out of the top 5 machines in TOP500 were built using GPGPUs as accelerators, as of January 2012. Despite such rapid evolution, we are facing yet another revolutionary change: an integrated GPU, i.e. CPUs and GPUs are integrated into the same package or even into the same die. In this paper, we are interested in how to utilize the integrated GPU to accelerate conventional operations (i.e., bulk memory operations such as memcpy or memset) that are normally performed by the CPUs, thus to improve application and overall system performance and resource utilizations without changing or rebuilding the application codes. In particular, we are interested in improving the performance of workloads that run on the server machines of commercial data centers. Besides computation requirements, these applications typically process a large volume of data, and devote large percentage of their processing time to I/O and manipulating the data.

2. ARCHITECTURE AND DESIGN

Our baseline architecture is a heterogeneous platform that has multiple CPU cores and GPU cores on a chip as shown in Figure 1. These heterogeneous processing units have different characteristics. First, a CPU core is a traditional, superscalar out-of-order core that is optimized for reducing the latency of each instruction through many architectural techniques such as branch prediction, data forwarding, and speculative instruction scheduling. Such a CPU core has a private instruction L1 cache and a private data L1 cache.

On the other hand, a GPU like SIMD core is a highly-multithreaded, wide SIMD core that is optimized for improving the overall throughput rather than latency. As a result, it does not support fancy speculation techniques. Rather, it is designed to consume less power and less area such as a multi-banked register file (rather than a multi-ported register file of an out-of-order core) to improve the overall throughput under a given power and area budget. In particular, a GPU core groups a large number of threads (for example, 256 threads) into one scheduling unit, termed a "warp", and executes the same instruction over those threads simultaneously. Furthermore, because a GPU core supports heavy multithreading among those warps and because it supports many more relaxed consistency model, it can support much more in-flight memory instructions than a CPU core, which goes through a private instruction and data L1 cache as shown in Figure 1. Note that different threads of a warp can issue non-coalesced memory requests. Our GPU L1 data cache models performs memory coalescing.

In our baseline, CPU cores and GPU cores share several

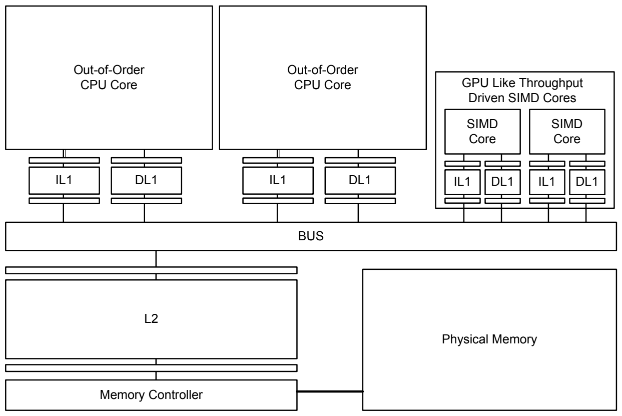

Figure 1: CPU Integrated with Throughput Driven GPU Like SIMD Cores

on-die resources. First, private L1 caches of both CPU and GPU cores are connected through an on-die bus sharing its bus bandwidth. Second, the on-die shared bus is connected to a large, shared L2 cache as shown in Figure 1. Lastly, CPU and GPU memory requests that miss in the L2 cache as well as dirty cache lines that need to be written-back from the L2 cache go through a shared memory controller. In addition to such shared resources, both CPU and GPU share the off-chip bus bandwidth and main memory space.

After bulk memory operation functions are called, the GPU aware memory copy function can decide whether the operations should be offloaded to the GPU cores. Often this decision can be based on the size of data movement. When deciding that the data movement operations should be offloaded, the GPU aware library will parallelize the data movement by creating multiple independent GPU threads and dispatch them to the integrated GPU cores. Compared with the CPU threads, GPU threads are much more lightweight and can be created with very little overhead. Furthermore, this process can be accelerated using a hardware engine that takes size, source and destination addresses of the data movement, and automatically creates the parallel GPU threads.

3. EXPERIMENTS AND ANALYSIS

To evaluate our solution, we used a functional full system emulator, Simics [2], combined with a cycle based architecture simulator that can model a heterogenous multi-core. FeS2 [1] is a timing-first, multiprocessor, x86 simulator, implemented as a module for Simics. MV5 [3] is a cycle based architecture simulator for heterogenous multi-core such as a processor integrating OOO CPU cores with large number of GPU like array-style SIMD cores. We combined MV5 with FeS2 where the GPU is simulated using MV5's infrastructure and the x86 workload is simulated by FeS2. In full system simulation, we apply the described bulk memory operation offloading to the GNU libc and the kernel (Linux kernel version 2.6.32.31).

As shown in Figure 2, for memory movement micro benchmarks, the speedup over CPU can be as much as 4.3 times depending on the workload sizes and the GPU configurations.

As shown in Figure 3, evaluation based on eight popular CPU benchmark applications with real-world input dataset shows that we can significantly improve the overall performance of those applications. When offload is combined with GPU caching and asynchronous support, the speedup is more than 30% for five, and more than 20% for two of the eight real world applications.

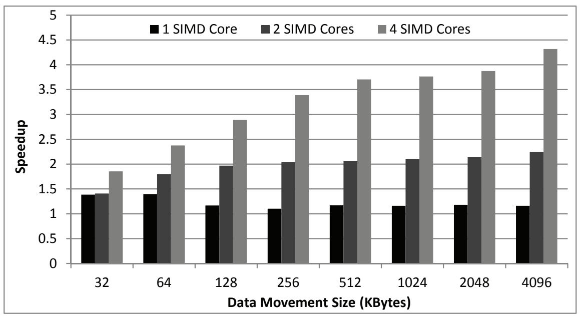

Figure 2: Speedup with Number of Cores for Different Data Movement Sizes

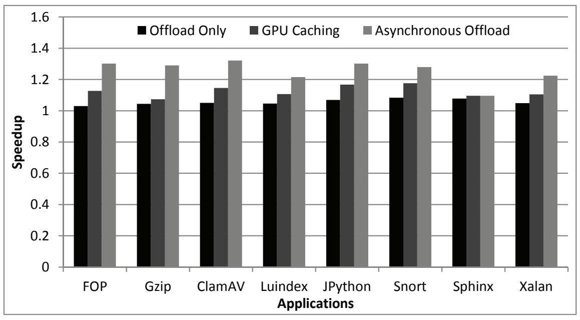

Figure 3: Speedup of Offloading Bulk Memory Operations to the Integrated GPU like Cores

4. CONCLUSIONS

In this paper, we proposed and presented a novel architectural support to improve the bulk memory operation performance for the CPU applications in a CPU-GPU heterogenous multicore processor. With very lightweight system and architectural support, our solution can take advantage of the most likely idle on-die GPU to improve the performance of bulk memory operations.

5. ADDITIONAL AUTHORS

Additional authors: Yonghong Yan (University of Houston, email: `yanyh@cs.uh.edu`) and Kyeong-An Kwon (University of Houston, email: `kyeongan@cs.uh.edu`).

6. REFERENCES

[1] Fes2: A full-system execution-driven simulator for x86. `http://fes2.cs.uiuc.edu/index.html`, 2007.

[2] Magnusson, P., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., and Werner, B. Simics: A full system simulation platform. *Computer 35*, 2 (Feb 2002), 50 –58.

[3] Meng, J., and Skadron, K. Avoiding cache thrashing due to private data placement in last-level cache for manycore scaling. In *Proceedings of the 2009 IEEE international conference on Computer design* (Piscataway, NJ, USA, 2009), ICCD'09, IEEE Press, pp. 282–288.